Savitch's Theorem

Theorem 25 (Savitch (1970))

$$\text{REACHABILITY} \in \text{SPACE}(\log^2 n).$$

- Let $G(V, E)$ be a graph with n nodes.
- For $i \geq 0$, let
  $$\text{PATH}(x, y, i)$$
  mean there is a path from node x to node y of length at most 2^i.
- There is a path from x to y if and only if
  $$\text{PATH}(x, y, \lceil \log n \rceil)$$
  holds.

The Proof (continued)

- For $i > 0$, PATH(x, y, i) if and only if there exists a z such that PATH$(x, z, i - 1)$ and PATH$(z, y, i - 1)$.
- For PATH$(x, y, 0)$, check the input graph or if $x = y$.
- Compute PATH$(x, y, \lceil \log n \rceil)$ with a depth-first search on a graph with nodes (x, y, z, i)s (see next page).[a]
- Like stacks in recursive calls, we keep only the current path of (x, y, i)s.
- The space requirement is proportional to the depth of the tree ($\lceil \log n \rceil$) times the size of the items stored at each node.

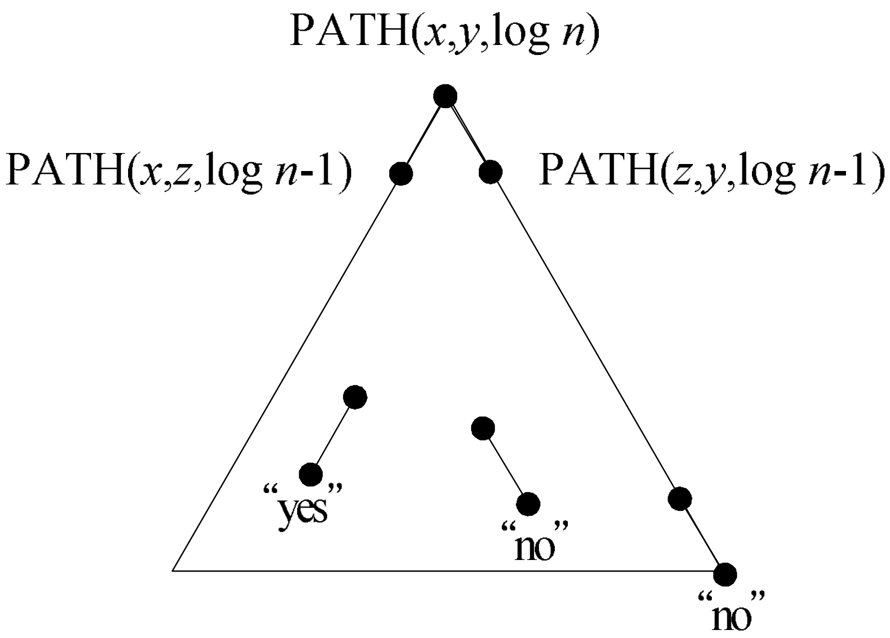
---

[a] Contributed by Mr. Chuan-Yao Tan on October 11, 2011.

The Proof (continued): Algorithm for PATH(x, y, i)

```
if $i = 0$ then
    if $x = y$ or $(x, y) \in E$ then
        return true;
    else
        return false;
    end if
else
    for $z = 1, 2, \ldots, n$ do
        if PATH$(x, z, i - 1)$ and PATH$(z, y, i - 1)$ then
            return true;
        end if
    end for
    return false;
end if
```

The Proof (continued)



The Proof (concluded)

- Depth is $\lceil \log n \rceil$, and each node (x, y, z, i) needs space $O(\log n)$.
- The total space is $O(\log^2 n)$.

The Relation between Nondeterministic Space and Deterministic Space Only Quadratic

Corollary 26 *Let* $f(n) \geq \log n$ *be proper. Then*

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)).$$

- Apply Savitch's proof to the configuration graph of the NTM on the input.
- From p. 242, the configuration graph has $O(c^{f(n)})$ nodes; hence each node takes space $O(f(n))$.
- But if we construct *explicitly* the whole graph before applying Savitch's theorem, we get $O(c^{f(n)})$ space!

The Proof (continued)

- The way out is *not* to generate the graph at all.
- Instead, keep the graph implicit.
- In fact, we check node connectedness only when $i = 0$ on p. 250, by examining the input string G.
- There, given configurations x and y, we go over the Turing machine's program to determine if there is an instruction that can turn x into y in one step.[a]

---

[a] Thanks to a lively class discussion on October 15, 2003.

The Proof (concluded)

- The z variable in the algorithm on p. 250 simply runs through all possible valid configurations.
  - Let $z = 0, 1, \ldots, O(c^{f(n)})$.
  - Make sure z is a valid configuration before using it in the recursive calls.[a]
- Each z has length $O(f(n))$ by Eq. (2) on p. 242.
- So each node needs space $O(f(n))$.
- The depth of the recursive call on p. 250 is $O(\log c^{f(n)})$, which is $O(f(n))$.
- The total space is therefore $O(f^2(n))$.

---

[a] Thanks to a lively class discussion on October 13, 2004.

Implications of Savitch's Theorem

- PSPACE = NPSPACE.
- Nondeterminism is less powerful with respect to space.
- Nondeterminism may be very powerful with respect to time as it is not known if P = NP.

Nondeterministic Space Is Closed under Complement

- Closure under complement is trivially true for deterministic complexity classes (p. 227).
- It is known that[a]

$$\text{coNSPACE}(f(n)) = \text{NSPACE}(f(n)). \tag{3}$$

- So

$$\begin{aligned} \text{coNL} &= \text{NL}, \\ \text{coNPSPACE} &= \text{NPSPACE}. \end{aligned}$$

- But it is not known whether coNP $=$ NP.

---

[a] Szelepscényi (1987) and Immerman (1988).

Reductions and Completeness

It is unworthy of excellent men
to lose hours like slaves in the labor of
computation.
— Gottfried Wilhelm von Leibniz (1646–1716)

Degrees of Difficulty

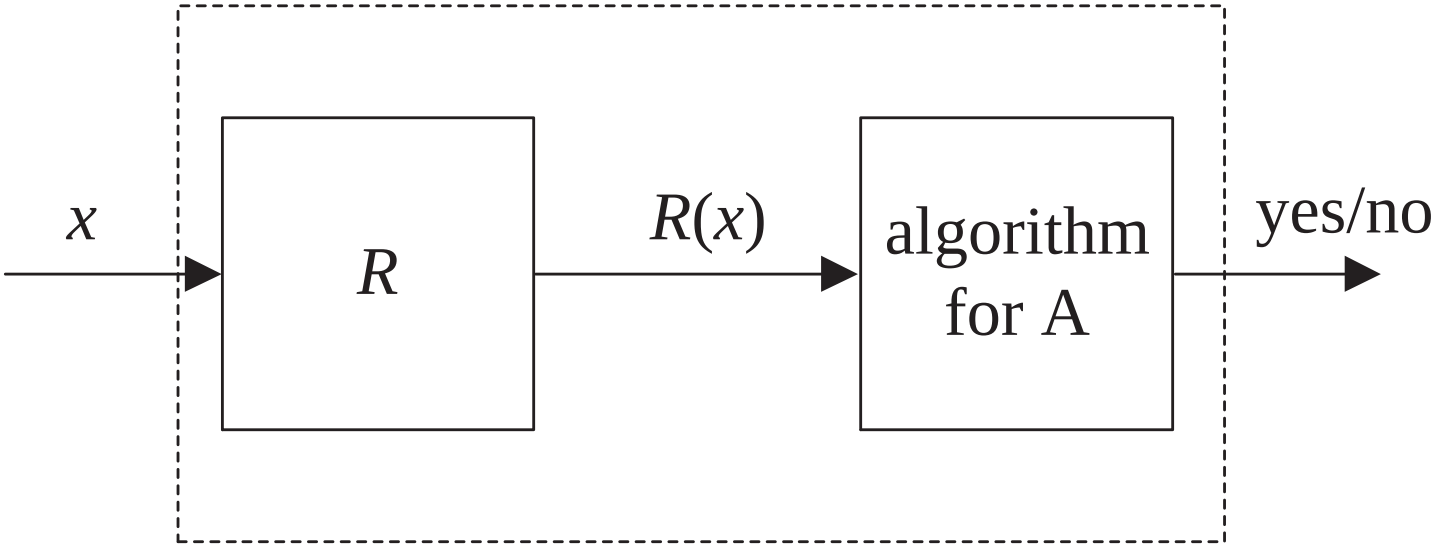
- When is a problem more difficult than another?
- B **reduces to** A if there is a transformation R which for every input x of B yields an input $R(x)$ of A.[a]
  - The answer to x for B is the same as the answer to $R(x)$ for A.
  - R is easy to compute.
- We say problem A is at least as hard as[b] problem B if B reduces to A.

---

[a] See also p. 164.

[b] Or simply "harder than" for brevity.

Reduction



Solving problem B by calling the algorithm for problem A *once* and *without* further processing its answer.

Degrees of Difficulty (concluded)

- This makes intuitive sense: If A is able to solve your problem B after only a little bit of work of R, then A must be at least as hard.
  - If A is easy to solve, it combined with R (which is also easy) would make B easy to solve, too.[a]
  - So if B is hard to solve, A must be hard (if not harder), too!

---

[a] Thanks to a lively class discussion on October 13, 2009.

Comments[a]

- Suppose B reduces to A via a transformation R.
- The input x is an instance of B.
- The output $R(x)$ is an instance of A.
- $R(x)$ may not span all possible instances of A.[b]
  - Some instances of A may never appear in the range of R.
- But x must be a general instance for B.

---

[a] Contributed by Mr. Ming-Feng Tsai (`D92922003`) on October 29, 2003.

[b] $R(x)$ may not be onto; Mr. Alexandr Simak (`D98922040`) on October 13, 2009.

Is "Reduction" a Confusing Choice of Word?[a]

- If B reduces to A, doesn't that intuitively make A smaller and simpler?
  - Sometimes, we say, "B can be reduced to A."
- But our definition means just the opposite.
- Our definition says in this case B is a special case of A.
- Hence A is harder.

---

[a] Moore and Mertens (2011).

Reduction between Languages

- Language L_1 is **reducible to** L_2 if there is a function R computable by a deterministic TM in space $O(\log n)$.
- Furthermore, for all inputs x, $x \in L_1$ if and only if $R(x) \in L_2$.
- R is said to be a (**Karp**) **reduction** from L_1 to L_2.

Reduction between Languages (concluded)

- Note that by Theorem 24 (p. 239), R runs in polynomial time.
  - In most cases, a polynomial-time R suffices for proofs.[a]
- Suppose R is a reduction from L_1 to L_2.
- Then solving "$R(x) \in L_2$?" is an algorithm for solving "$x \in L_1$?"[b]

---

[a] In fact, unless stated otherwise, we will only require that the reduction R run in polynomial time.

[b] Of course, it may not be an optimal one.

A Paradox?

- Degree of difficulty is not defined in terms of *absolute* complexity.
- So a language B $\in$ TIME(n^{99}) may be "easier" than a language A $\in$ TIME(n^3).
  - Again, this happens when B is reducible to A.
- But isn't this a contradiction if the best algorithm for B requires n^{99} steps?
- That is, how can a problem *requiring* n^{99} steps be reducible to a problem solvable in n^3 steps?

Paradox Resolved

- The so-called contradiction does not hold.
- Suppose we solve the problem "$x \in$ B?" via "$R(x) \in$ A?"
- We must consider the time spent by $R(x)$ and its length $|\, R(x) \,|$:
  - Because $R(x)$ (not x) is solved by A.

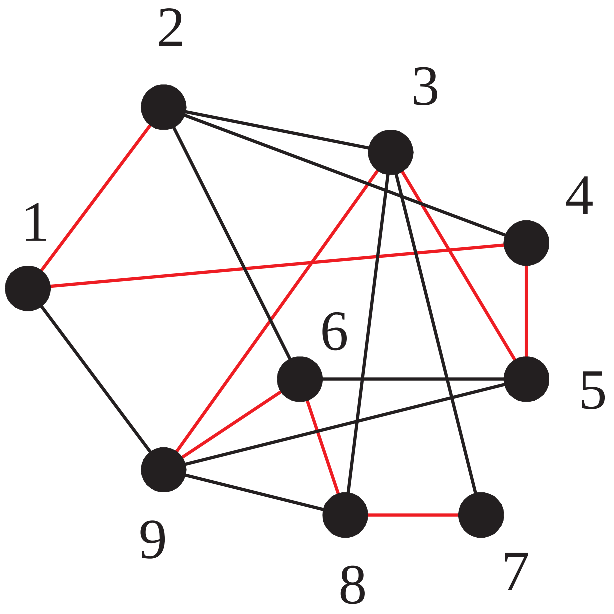
HAMILTONIAN PATH

- A **Hamiltonian path** of a graph is a path that visits every node of the graph exactly once.
- Suppose graph G has n nodes: $1, 2, \ldots, n$.
- A Hamiltonian path can be expressed as a permutation π of $\{ 1, 2, \ldots, n \}$ such that
  - $\pi(i) = j$ means the ith position is occupied by node j.
  - $(\pi(i), \pi(i + 1)) \in G$ for $i = 1, 2, \ldots, n - 1$.
- HAMILTONIAN PATH asks if a graph has a Hamiltonian path.

Reduction of HAMILTONIAN PATH to SAT

- Given a graph G, we shall construct a CNF $R(G)$ such that $R(G)$ is satisfiable iff G has a Hamiltonian path.
- $R(G)$ has n^2 boolean variables x_{ij}, $1 \leq i, j \leq n$.
- x_{ij} means

  "the ith position in the Hamiltonian path is occupied by node j."
- Our reduction will produce clauses.



$x_{12} = x_{21} = x_{34} = x_{45} = x_{53} = x_{69} = x_{76} = x_{88} = x_{97} = 1$; $\pi(1) = 2, \pi(2) = 1, \pi(3) = 4, \pi(4) = 5, \pi(5) = 3, \pi(6) = 9, \pi(7) = 6, \pi(8) = 8, \pi(9) = 7$.

The Clauses of $R(G)$ and Their Intended Meanings

1. Each node j must appear in the path.
   - $x_{1j} \lor x_{2j} \lor \cdots \lor x_{nj}$ for each j.
2. No node j appears twice in the path.
   - $\neg x_{ij} \lor \neg x_{kj} (\equiv \neg (x_{ij} \land x_{kj}))$ for all i, j, k with $i \neq k$.
3. Every position i on the path must be occupied.
   - $x_{i1} \lor x_{i2} \lor \cdots \lor x_{in}$ for each i.
4. No two nodes j and k occupy the same position in the path.
   - $\neg x_{ij} \lor \neg x_{ik} (\equiv \neg (x_{ij} \land x_{ik}))$ for all i, j, k with $j \neq k$.
5. Nonadjacent nodes i and j cannot be adjacent in the path.
   - $\neg x_{ki} \lor \neg x_{k+1,j}$ for all $(i, j) \not\in G$ and $k = 1, 2, \ldots, n-1$.

The Proof

- $R(G)$ contains $O(n^3)$ clauses.
- $R(G)$ can be computed efficiently (simple exercise).
- Suppose $T \models R(G)$.
- From the 1st and 2nd types of clauses, for each node j there is a unique position i such that $T \models x_{ij}$.
- From the 3rd and 4th types of clauses, for each position i there is a unique node j such that $T \models x_{ij}$.
- So there is a permutation π of the nodes such that $\pi(i) = j$ if and only if $T \models x_{ij}$.

The Proof (concluded)

- The 5th type of clauses furthermore guarantee that $(\pi(1), \pi(2), \ldots, \pi(n))$ is a Hamiltonian path.
- Conversely, suppose G has a Hamiltonian path
$$(\pi(1), \pi(2), \ldots, \pi(n)),$$
where π is a permutation.
- Clearly, the truth assignment
$$T(x_{ij}) = \texttt{true} \text{ if and only if } \pi(i) = j$$
satisfies all clauses of $R(G)$.

A Comment[a]

- An answer to "Is $R(G)$ satisfiable?" does answer "Is G Hamiltonian?"
- But a positive answer does not give a Hamiltonian path for G.
  - Providing a witness is not a requirement of reduction.
- A positive answer to "Is $R(G)$ satisfiable?" *plus a satisfying truth assignment* does provide us with a Hamiltonian path for G.

---

[a] Contributed by Ms. Amy Liu (`J94922016`) on May 29, 2006.

Reduction of REACHABILITY to CIRCUIT VALUE

- Note that both problems are in P.
- Given a graph $G = (V, E)$, we shall construct a *variable-free* circuit $R(G)$.
- The output of $R(G)$ is true if and only if there is a path from node 1 to node n in G.
- Idea: the Floyd-Warshall algorithm.

The Gates

- The gates are
  - g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$.
  - h_{ijk} with $1 \leq i, j, k \leq n$.
- g_{ijk}: There is a path from node i to node j without passing through a node bigger than k.
- h_{ijk}: There is a path from node i to node j passing through k but not any node bigger than k.
- Input gate $g_{ij0} = \texttt{true}$ if and only if $i = j$ or $(i, j) \in E$.

The Construction

- h_{ijk} is an AND gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$, where $k = 1, 2, \ldots, n$.
- g_{ijk} is an OR gate with predecessors $g_{i,j,k-1}$ and $h_{i,j,k}$, where $k = 1, 2, \ldots, n$.
- g_{1nn} is the output gate.
- Interestingly, $R(G)$ uses no $\neg$ gates.
  - It is a **monotone circuit**.

Reduction of CIRCUIT SAT to SAT

- Given a circuit C, we will construct a boolean expression $R(C)$ such that $R(C)$ is satisfiable iff C is.
  - $R(C)$ will turn out to be a CNF.
  - $R(C)$ is basically a depth-2 circuit; furthermore, each gate has out-degree 1.
- The variables of $R(C)$ are those of C plus g for each gate g of C.
  - The g's propagate the truth values for the CNF.
- Each gate of C will be turned into equivalent clauses.
- Recall that clauses are $\wedge$ed together by definition.

The Clauses of $R(C)$

g **is a variable gate** x**:** Add clauses $(\neg g \lor x)$ and $(g \lor \neg x)$.

- Meaning: $g \Leftrightarrow x$.

g **is a `true` gate:** Add clause (g).

- Meaning: g must be true to make $R(C)$ true.

g **is a `false` gate:** Add clause $(\neg g)$.

- Meaning: g must be false to make $R(C)$ true.

g **is a $\neg$ gate with predecessor gate** h**:** Add clauses $(\neg g \lor \neg h)$ and $(g \lor h)$.

- Meaning: $g \Leftrightarrow \neg h$.

The Clauses of $R(C)$ (concluded)

g **is a** $\vee$ **gate with predecessor gates** h **and** h'**:** Add clauses $(\neg h \vee g)$, $(\neg h' \vee g)$, and $(h \vee h' \vee \neg g)$.

- Meaning: $g \Leftrightarrow (h \vee h')$.

g **is a** $\wedge$ **gate with predecessor gates** h **and** h'**:** Add clauses $(\neg g \vee h)$, $(\neg g \vee h')$, and $(\neg h \vee \neg h' \vee g)$.

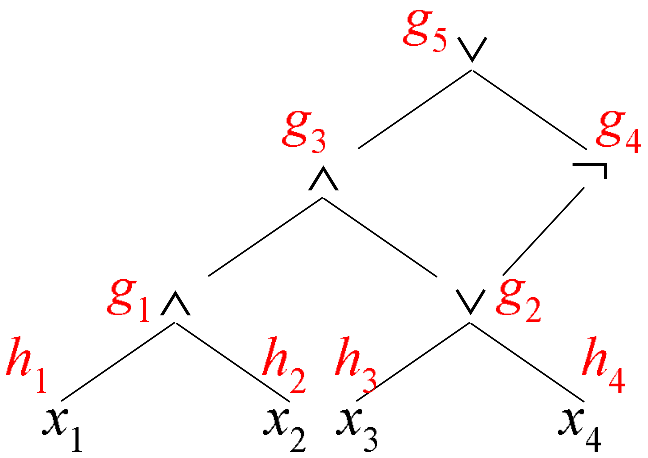
- Meaning: $g \Leftrightarrow (h \wedge h')$.

g **is the output gate:** Add clause (g).

- Meaning: g must be true to make $R(C)$ true.

Note: If gate g feeds gates $h_1, h_2, \ldots$, then variable g appears in the clauses for $h_1, h_2, \ldots$ in $R(C)$.

An Example



$$\begin{aligned}
& (h_1 \Leftrightarrow x_1) \wedge (h_2 \Leftrightarrow x_2) \wedge (h_3 \Leftrightarrow x_3) \wedge (h_4 \Leftrightarrow x_4) \\
\wedge \quad & [\, g_1 \Leftrightarrow (h_1 \wedge h_2)\,] \wedge [\, g_2 \Leftrightarrow (h_3 \vee h_4)\,] \\
\wedge \quad & [\, g_3 \Leftrightarrow (g_1 \wedge g_2)\,] \wedge (g_4 \Leftrightarrow \neg g_2) \\
\wedge \quad & [\, g_5 \Leftrightarrow (g_3 \vee g_4)\,] \wedge g_5 .
\end{aligned}$$

An Example (concluded)

- In general, the result is a CNF.
- The CNF has size proportional to the circuit's number of gates.
- The CNF adds new variables to the circuit's original input variables.
- Had we used the idea on p. 209 for the reduction, the resulting formula may have an exponential length because of the copying.[a]

---

[a] Contributed by Mr. Ching-Hua Yu (`D00921025`) on October 16, 2012.

Composition of Reductions

Proposition 27 *If* R_{12} *is a reduction from* L_1 *to* L_2 *and* R_{23} *is a reduction from* L_2 *to* L_3*, then the composition* $R_{12} \circ R_{23}$ *is a reduction from* L_1 *to* L_3*.*

- So reducibility is transitive.

Completeness[a]

- As reducibility is transitive, problems can be ordered with respect to their difficulty.
- Is there a *maximal* element (the *hardest* problem)?
- It is not obvious that there should be a maximal element.
  - Many infinite structures (such as integers and real numbers) do not have maximal elements.
- Hence it may surprise you that most of the complexity classes that we have seen so far have maximal elements.

---

[a] Cook (1971) and Levin (1973).

Completeness (concluded)

- Let $\mathcal{C}$ be a complexity class and $L \in \mathcal{C}$.
- L is **$\mathcal{C}$-complete** if every $L' \in \mathcal{C}$ can be reduced to L.
  - Most complexity classes we have seen so far have complete problems!
- Complete problems capture the difficulty of a class because they are the hardest problems in the class.

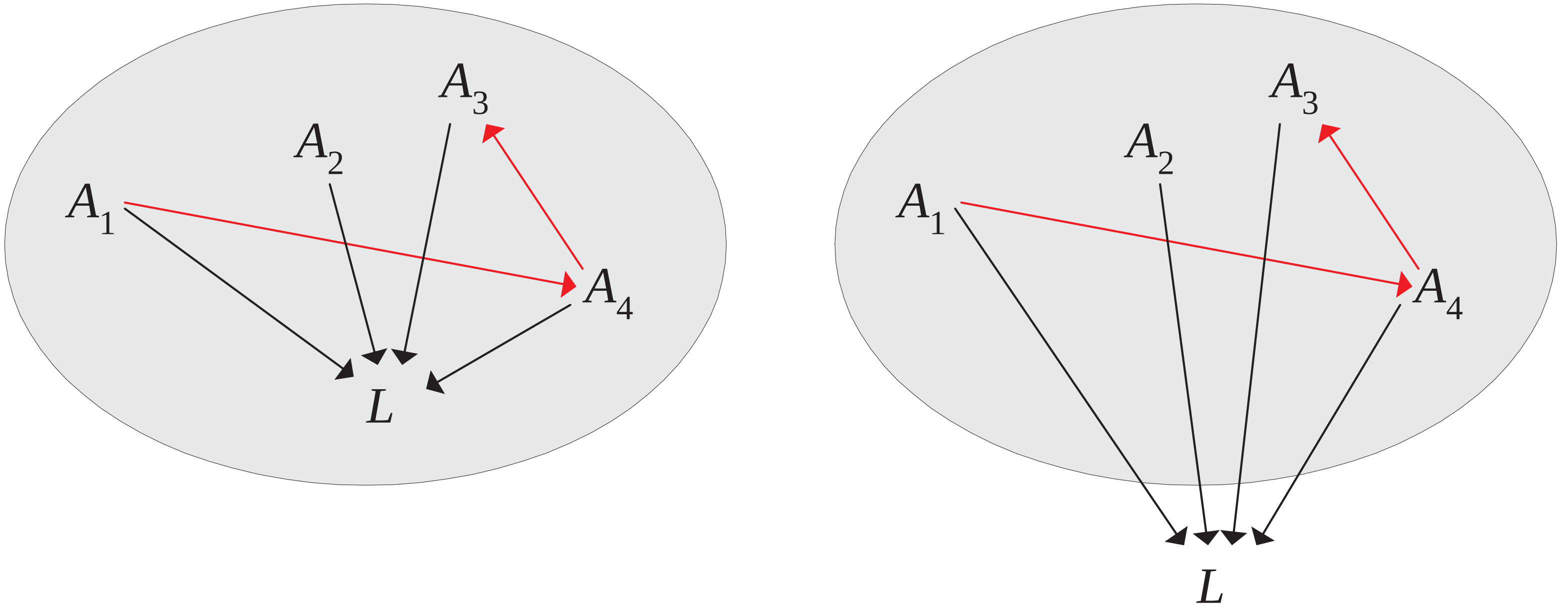
Hardness

- Let $\mathcal{C}$ be a complexity class.
- L is **$\mathcal{C}$-hard** if every $L' \in \mathcal{C}$ can be reduced to L.
- It is not required that $L \in \mathcal{C}$.
- If L is $\mathcal{C}$-hard, then by definition, every $\mathcal{C}$-complete problem can be reduced to L.[a]

---

[a] Contributed by Mr. Ming-Feng Tsai (`D92922003`) on October 15, 2003.

Illustration of Completeness and Hardness



Closedness under Reductions

- A class $\mathcal{C}$ is **closed under reductions** if whenever L is reducible to L' and $L' \in \mathcal{C}$, then $L \in \mathcal{C}$.
- It is easy to show that P, NP, coNP, L, NL, PSPACE, and EXP are all closed under reductions.

Complete Problems and Complexity Classes

Proposition 28 *Let $\mathcal{C}'$ and $\mathcal{C}$ be two complexity classes such that $\mathcal{C}' \subseteq \mathcal{C}$. Assume $\mathcal{C}'$ is closed under reductions and L is $\mathcal{C}$-complete. Then $\mathcal{C} = \mathcal{C}'$ if and only if $L \in \mathcal{C}'$.*

- Suppose $L \in \mathcal{C}'$ first.
- Every language $A \in \mathcal{C}$ reduces to $L \in \mathcal{C}'$.
- Because $\mathcal{C}'$ is closed under reductions, $A \in \mathcal{C}'$.
- Hence $\mathcal{C} \subseteq \mathcal{C}'$.
- As $\mathcal{C}' \subseteq \mathcal{C}$, we conclude that $\mathcal{C} = \mathcal{C}'$.

The Proof (concluded)

- On the other hand, suppose $\mathcal{C} = \mathcal{C}'$.
- As L is $\mathcal{C}$-complete, $L \in \mathcal{C}$.
- Thus, trivially, $L \in \mathcal{C}'$.

Two Important Corollaries

Proposition 28 implies the following.

Corollary 29 *$P = NP$ if and only if an NP-complete problem in P.*

Corollary 30 *$L = P$ if and only if a P-complete problem is in L.*

Complete Problems and Complexity Classes

Proposition 31 *Let $\mathcal{C}'$ and $\mathcal{C}$ be two complexity classes closed under reductions. If L is complete for both $\mathcal{C}$ and $\mathcal{C}'$, then $\mathcal{C} = \mathcal{C}'$.*

- All languages $\mathcal{L} \in \mathcal{C}$ reduce to $L \in \mathcal{C}$ and $L \in \mathcal{C}'$.
- Since $\mathcal{C}'$ is closed under reductions, $\mathcal{L} \in \mathcal{C}'$.
- Hence $\mathcal{C} \subseteq \mathcal{C}'$.
- The proof for $\mathcal{C}' \subseteq \mathcal{C}$ is symmetric.

Table of Computation

- Let $M = (K, \Sigma, \delta, s)$ be a single-string polynomial-time deterministic TM deciding L.
- Its computation on input x can be thought of as a $|\,x\,|^k \times |\,x\,|^k$ table, where $|\,x\,|^k$ is the time bound.
  - It is essentially a sequence of configurations.
- Rows correspond to time steps 0 to $|\,x\,|^k - 1$.
- Columns are positions in the string of M.
- The (i, j)th table entry represents the contents of position j of the string *after* i steps of computation.

Some Conventions To Simplify the Table

- M halts after at most $|\,x\,|^k - 2$ steps.
- Assume a large enough k to make it true for $|\,x\,| \geq 2$.
- Pad the table with $\sqcup$s so that each row has length $|\,x\,|^k$.
  - The computation will never reach the right end of the table for lack of time.
- If the cursor scans the jth position at time i when M is at state q and the symbol is σ, then the (i, j)th entry is a *new* symbol σ_q.

Some Conventions To Simplify the Table (continued)

- If q is "yes" or "no," simply use "yes" or "no" instead of σ_q.
- Modify M so that the cursor starts not at $\triangleright$ but at the first symbol of the input.
- The cursor never visits the leftmost $\triangleright$ by telescoping two moves of M each time the cursor is about to move to the leftmost $\triangleright$.
- So the first symbol in every row is a $\triangleright$ and not a $\triangleright_q$.

Some Conventions To Simplify the Table (concluded)

- Suppose M has halted before its time bound of $|\,x\,|^k$, so that “yes” or “no” appears at a row before the last.
- Then all subsequent rows will be identical to that row.
- M accepts x if and only if the $(|\,x\,|^k - 1, j)$th entry is “yes” for some position j.

Comments

- Each row is essentially a configuration.
- If the input $x = 010001$, then the first row is

$$\overbrace{\triangleright \mathtt{0}_{\mathtt{s}}\mathtt{10001} \sqcup \sqcup \cdots \sqcup}^{|\,x\,|^k}$$

- A typical row may look like

$$\overbrace{\triangleright \mathtt{10100}_{\mathtt{q}}\mathtt{01110100} \sqcup \sqcup \cdots \sqcup}^{|\,x\,|^k}$$

Comments (concluded)

- The last rows must look like

$$\overbrace{\triangleright \cdots \text{``yes''} \cdots \sqcup}^{|\,x\,|^k} \quad \text{or} \quad \overbrace{\triangleright \cdots \text{``no''} \cdots \sqcup}^{|\,x\,|^k}$$

- Three out of the table's 4 borders are known:

$$\begin{array}{cccccccc}
\triangleright & \text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} & \sqcup \\
\triangleright & & & & & & & \sqcup \\
\triangleright & & & & & & & \sqcup \\
\triangleright & & & & & & & \sqcup \\
\triangleright & & & & & & & \sqcup \\
 & & & & \vdots & & &
\end{array}$$